

# Machine Learning/Deep Learning on Summit

Junqi Yin

Advanced Data and Workflow

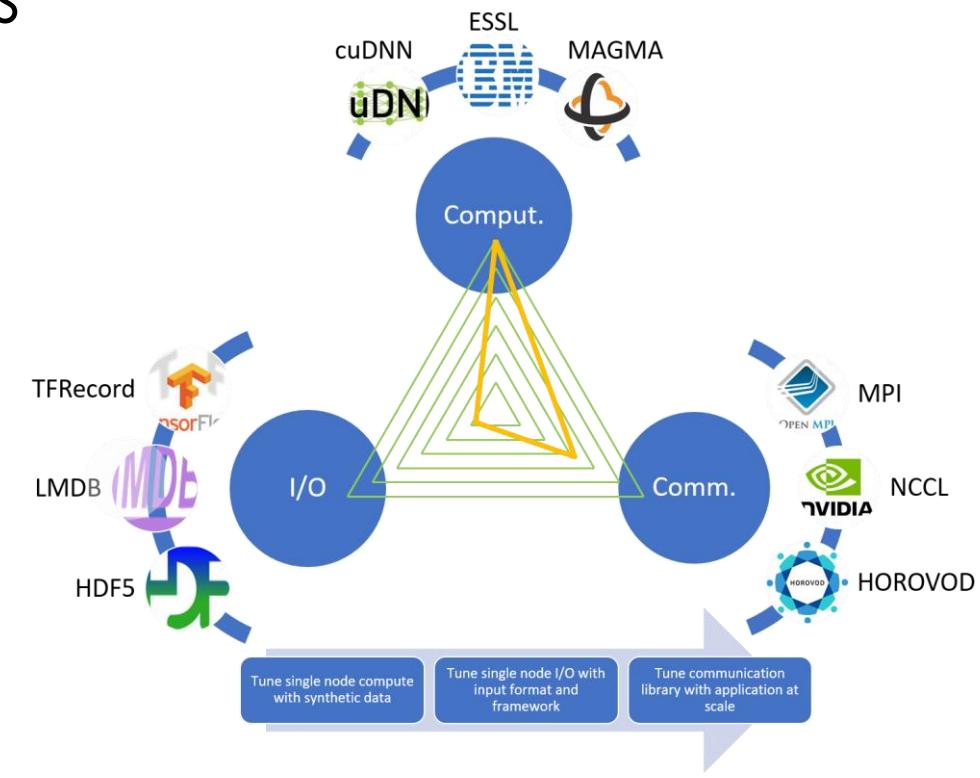
Oak Ridge Leadership Computing Facility



ORNL is managed by UT-Battelle, LLC for the US Department of Energy

# Outline

- Overview
- ML/DL software stack on Summit
- Distributed deployment considerations
- Performance baselines
  - ML: PCA, Kmeans, SVM, XGBoost
  - DL: image(ResNet), text (BERT)
- Scaling practices
  - Compute, I/O, Communication
- Conclusion



# ML/DL applications on Summit overview

- ML/DL has entered exascale computing
  - (1) “Exascale Deep Learning for Climate Analytics”
  - (2) “Exascale Deep Learning to Accelerate Cancer Research”
  - (3) “Exascale Deep Learning for Scientific Inverse Problems”

Application	Network	Sustained Performance (ExaFlops)	Peak Performance (ExaFlops)
(1) <u>Climate</u>	DeepLabV3+	0.999	1.13
(2) <u>Medical</u>	MENNDL	1.3	n/a
(3) <u>Materials</u>	Tiramisu variant	1.5	2.1

# ML/DL applications on Summit overview

- ML/DL by domains and methods (growing list)

Domain\Method	Supervised Learning					Unsupervised Learning		Reinforcement Learning		Hyperparameter Search		
	Random Forest	MLP	CNN	RNN (LSTM)	Auto Encoder	Clustering	Evolution	Transformer	Q-Learning	Agent-based Learning	Evolution	RL-based
Biophysics				✓	✓			✓	✓			
Chemistry		✓			✓							
Climate	✓	✓	✓		✓	✓						
Computer Science		✓	✓		✓		✓			✓	✓	✓
Fusion		✓										
Life Sciences	✓											
Material Sciences	✓	✓										
Medical Sciences		✓										
Nuclear Physics		✓	✓									
Turbulence		✓	✓									
Particle Physics		✓	✓		✓							

# ML/DL stack on Summit

```
module load ibm-wml-ce
```

Machine learning



**RAPIDS**

cuDF cuML

Dask, CuPy, XGBoost

Snap ML

Logistic Regression

Random Forest

Decision Tree

SVM

Ridge Regression

Lasso Regression

Frameworks

Deep learning



TensorFlow  
TensorFlow Probability  
TensorBoard  
TensorFlow-Keras



Plugins and Tools



IBM LMS



NVIDIA

DALI

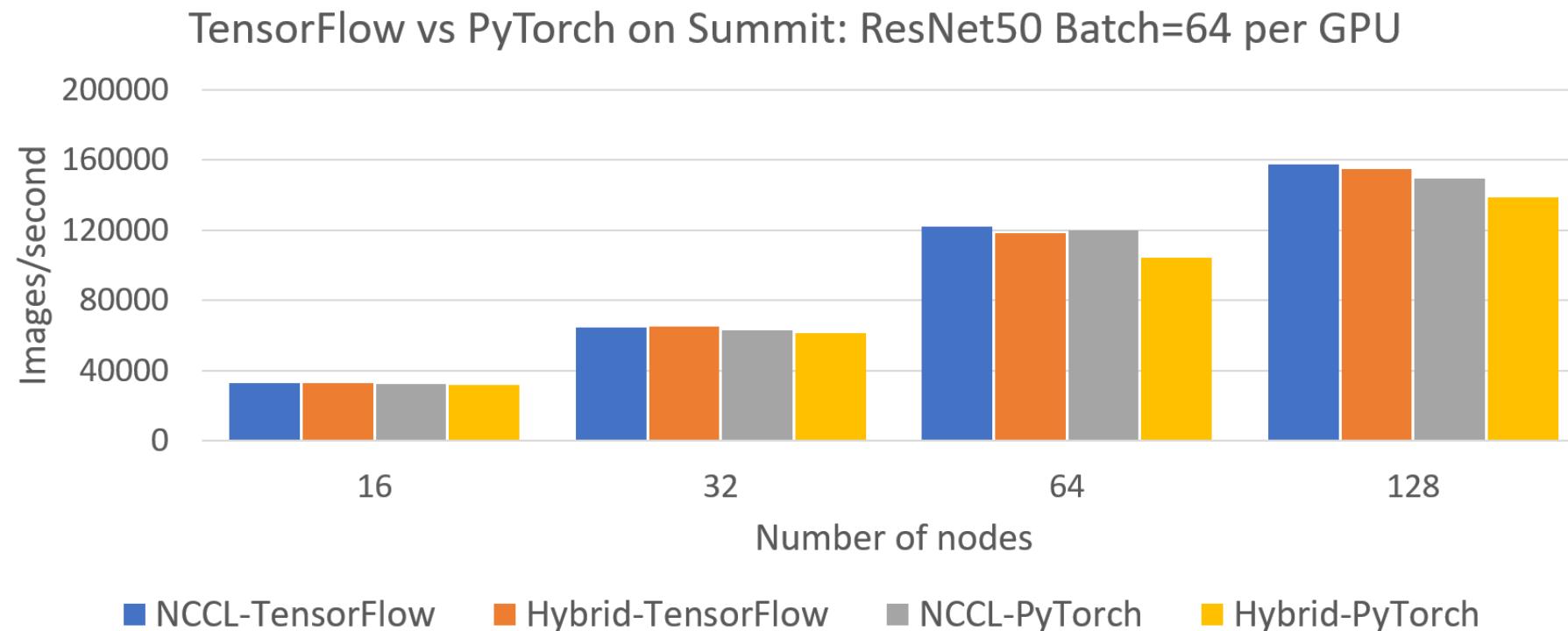
Apex

TensorRT

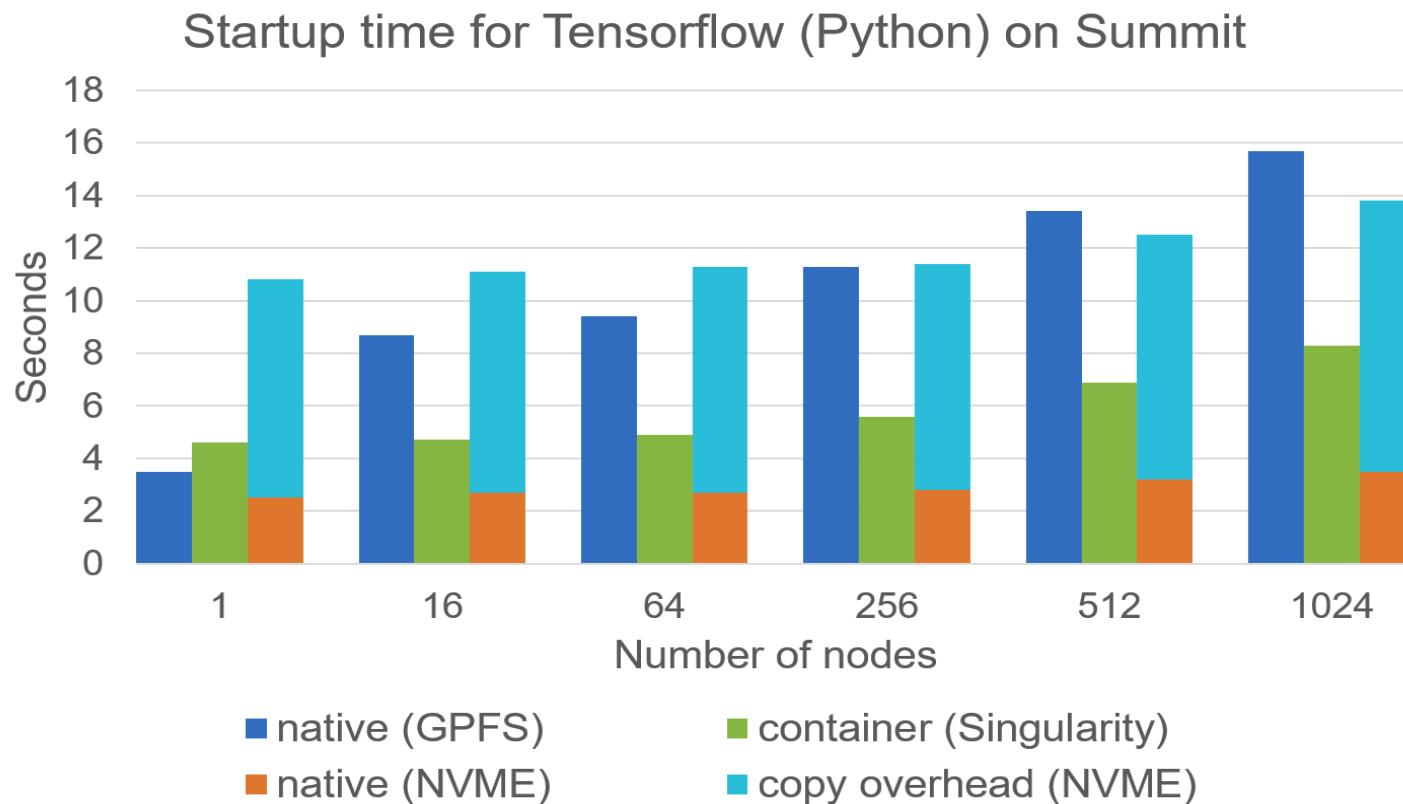
Visualization & Workflow: ParaView, Visit, SIGHT, Pegasus, Radical Toolkit, ...

# Distributed deployment considerations

- Framework consideration
  - TensorFlow vs PyTorch
  - NCCL vs MPI backend



# Distributed deployment considerations

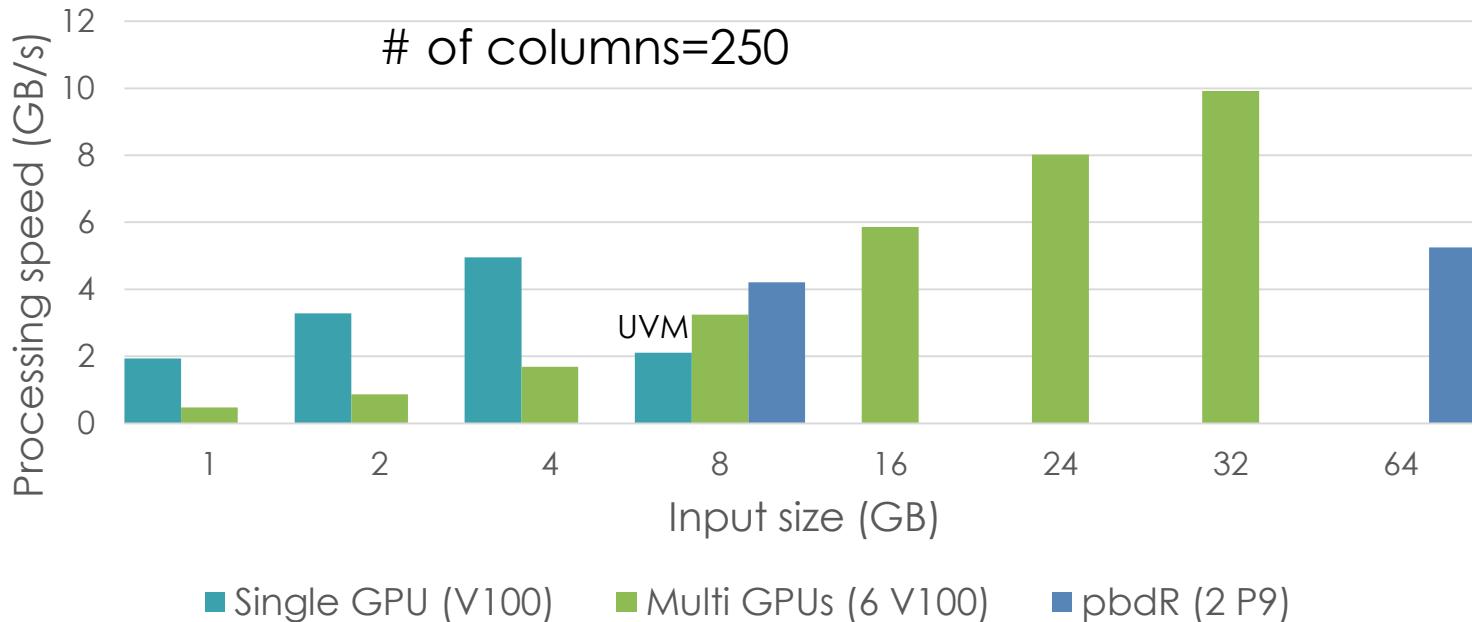


- Native vs Container
  - Impact of loading shared libs: manageable on GPFS
  - Runtime performance: comparable

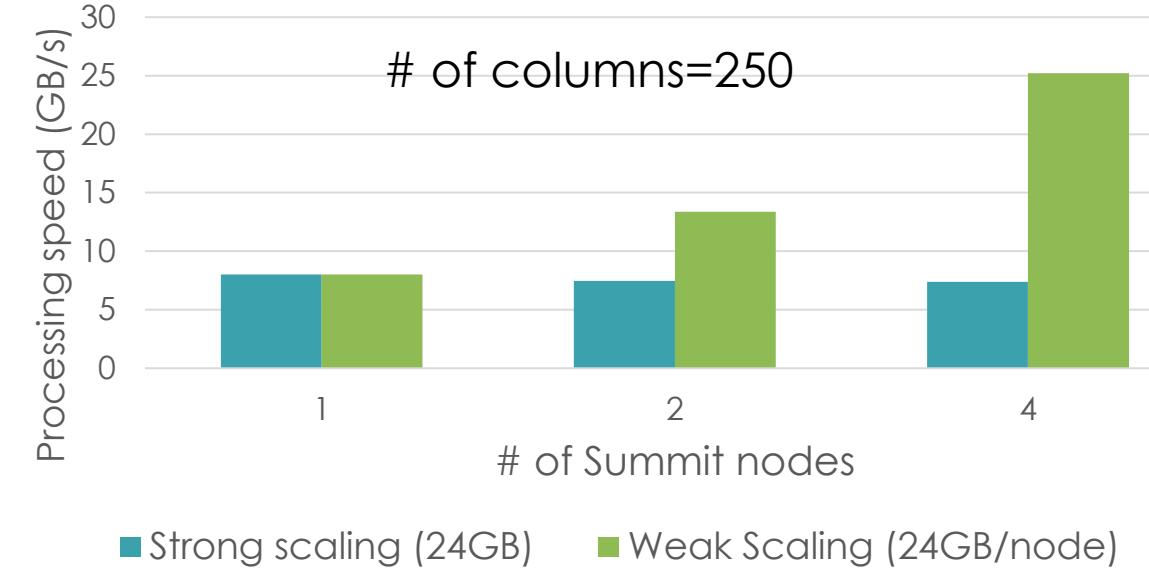
# Performance baselines: PCA

- Tall skinny input
- Single node
  - Up to 10 GB/s (32 GB input)
  - Single GPU + UVM < Multi-GPU

cuML PCA on single Summit node  
# of columns=250

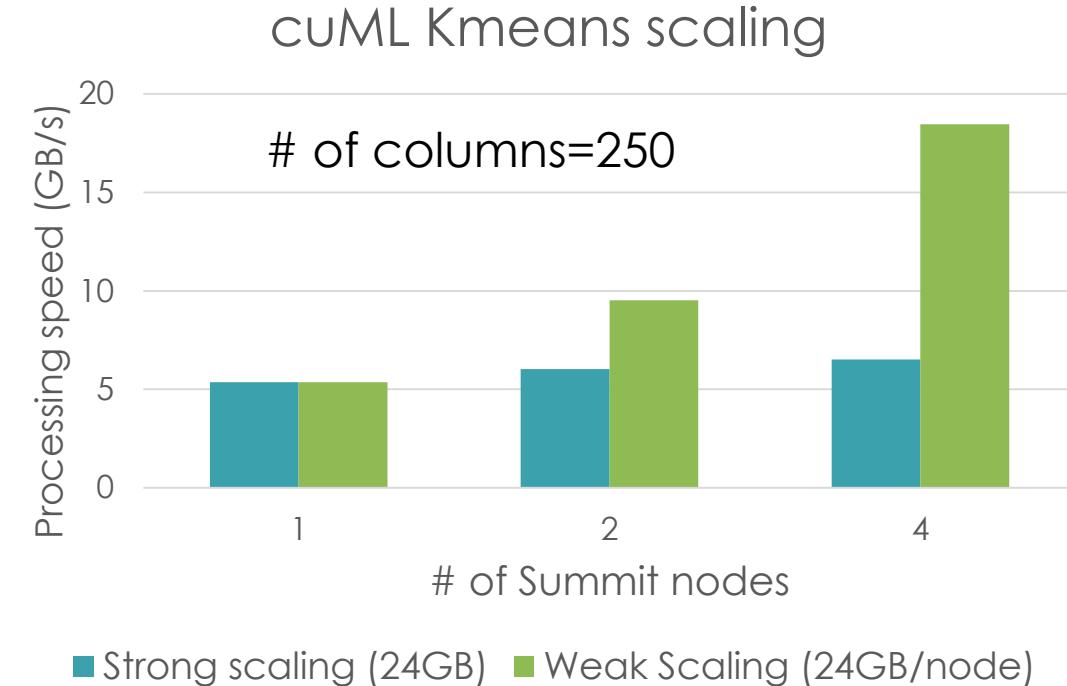
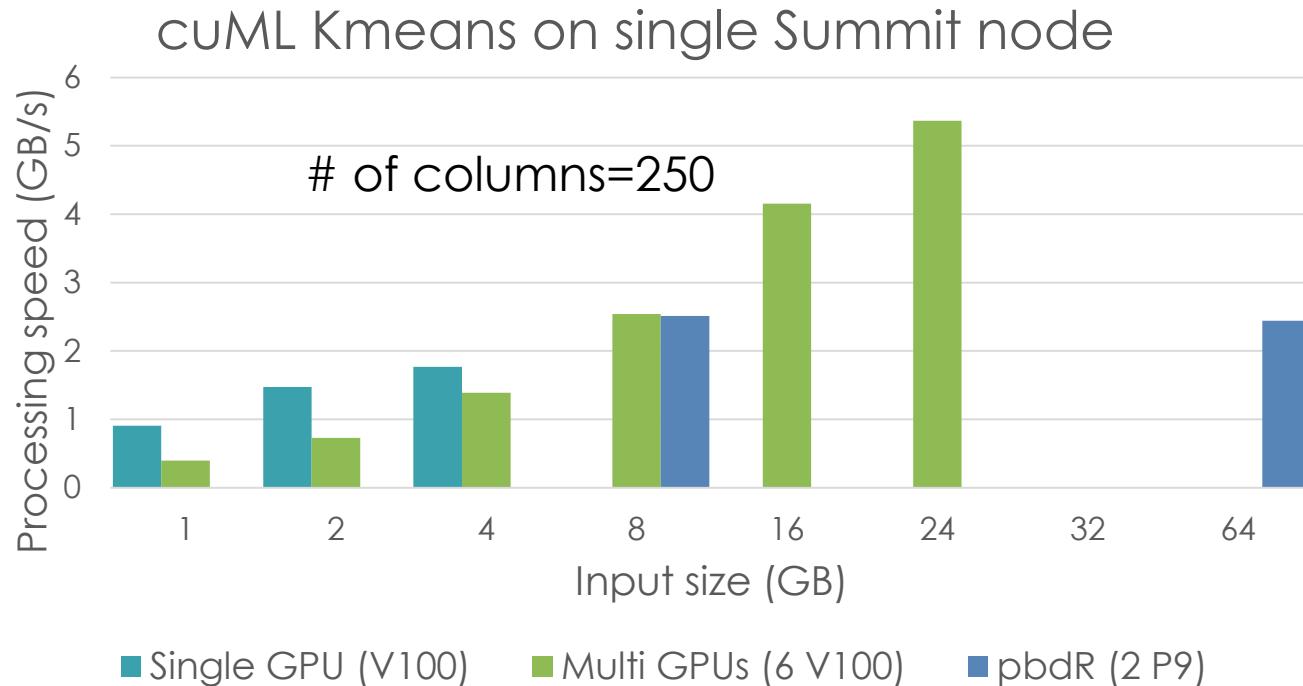


cuML PCA scaling



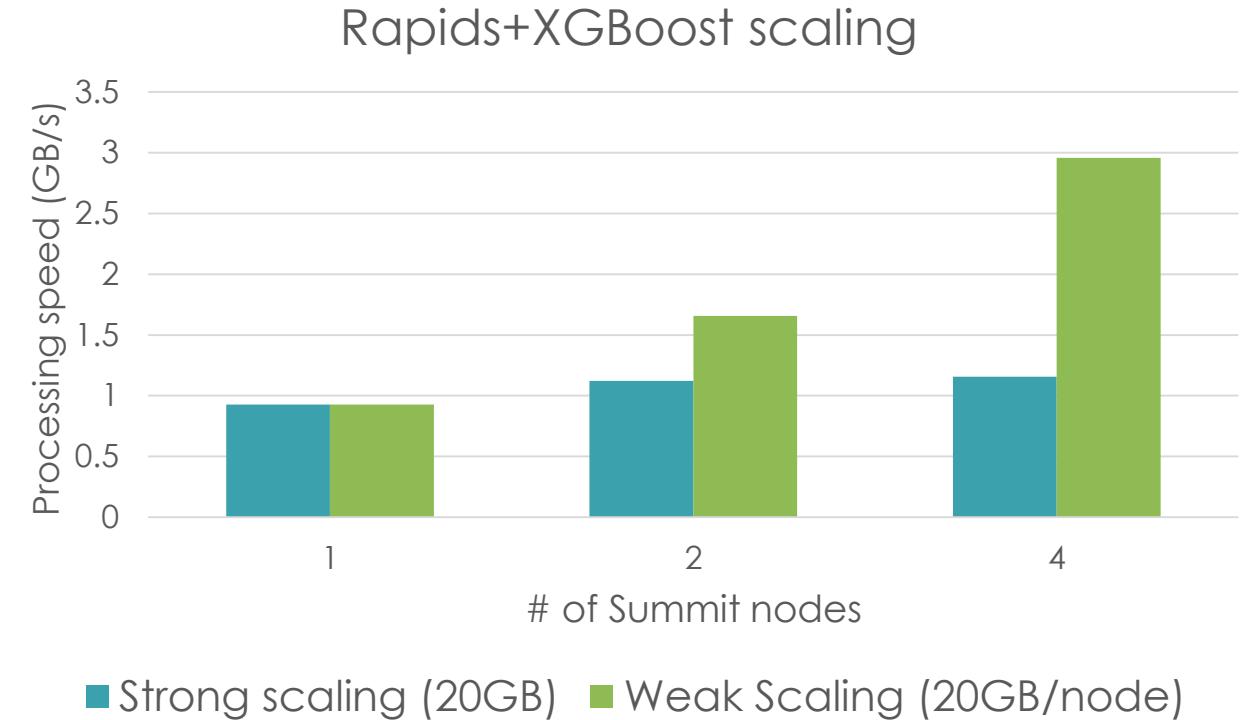
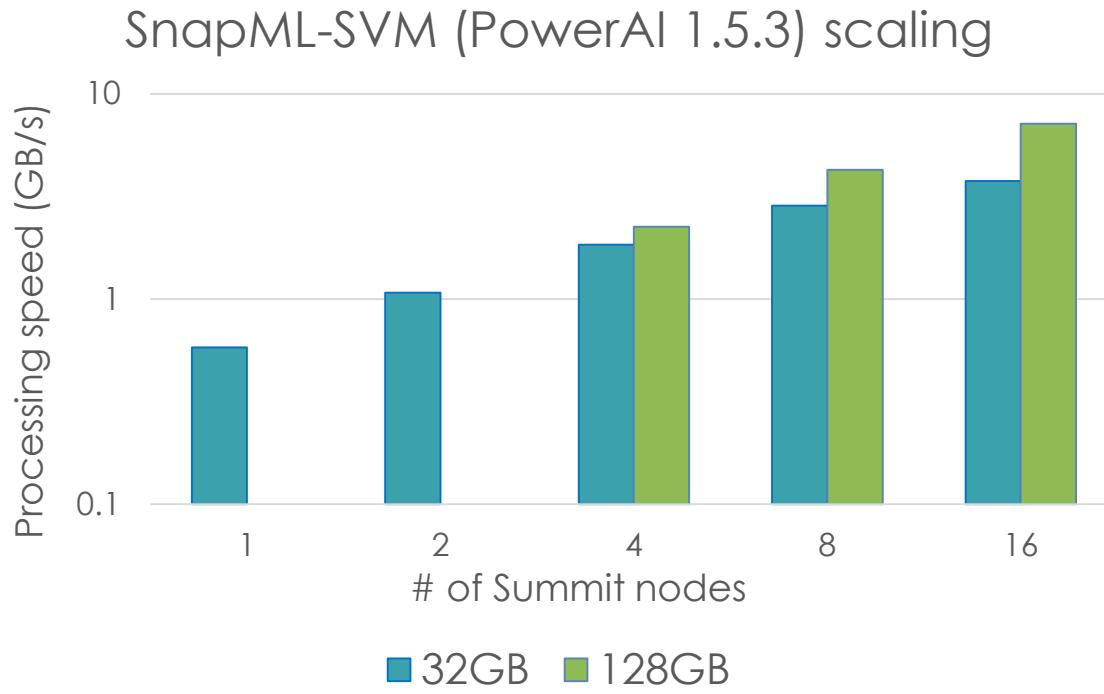
- Strong vs weak scaling
- Rapids is evolving

# Performance baselines: Kmeans



- Tall skinny input
- Single node: up to 5.4 GB/s (24 GB input)
- Strong vs weak scaling

# Performance baselines: SVM, XGBoost

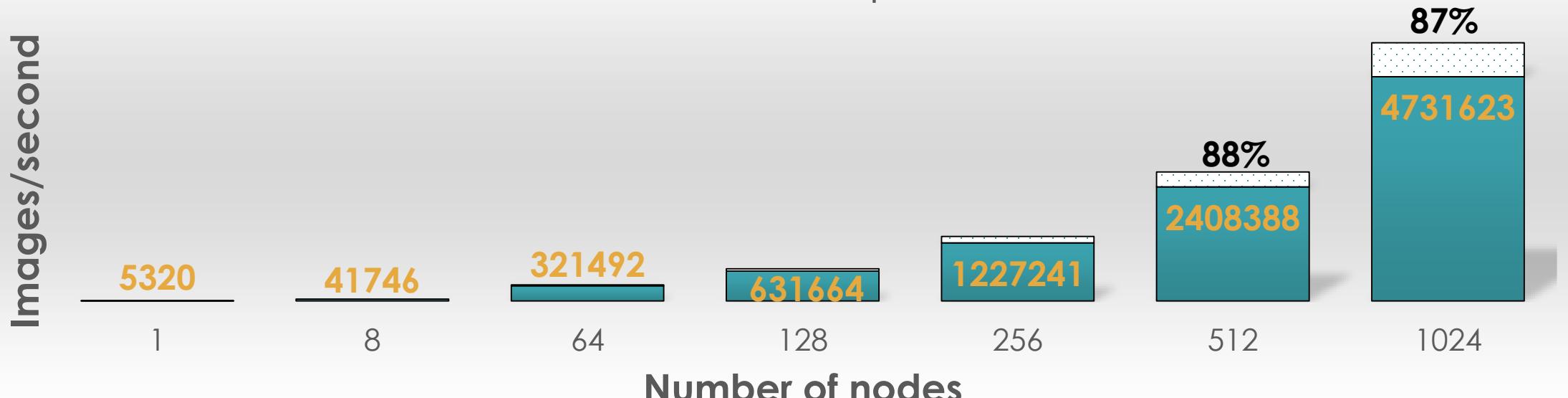


- SnapML SVM better strong scaling
- Rapids cuDF + Dask + XGBoost

[GTC20 Slides](#)

# Performance baselines: ResNet50 on ImageNet

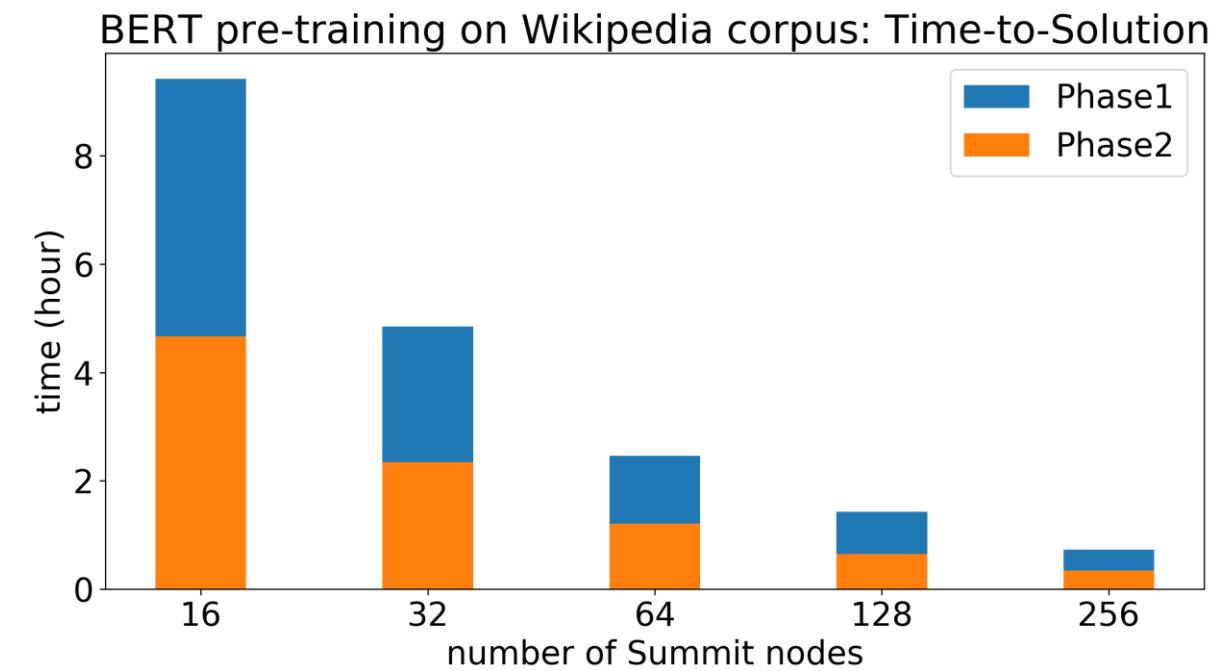
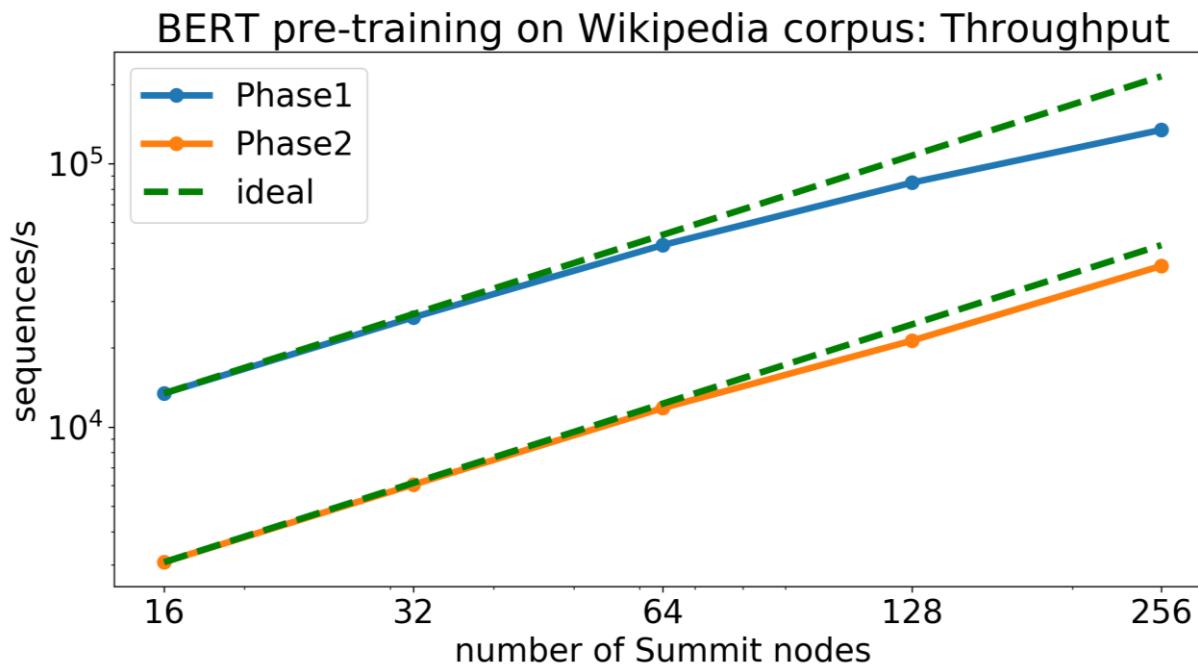
TF\_CNN\_Benchmark on Summit: ResNet50  
batch-size = 256 per GPU



Nodes	Mini-batch size	Top-1 Val accuracy	Training time (min)
16	12288	0.750	27
32	12288	0.766	17
64	15360	0.763	12

TF.distribute & Horovod + LARS  
[code: TensorFlow distributed example](#)

# Performance baselines: BERT on Wikipedia



- Throughput: 63% (phase1) 83% (phase2)
- Time-to-Solution: ~64min on 1536 V100 vs ~76min on 1024 TPUv3 ([arXiv:1904.00962](https://arxiv.org/abs/1904.00962))

[code: PyTorch BERT example](#)

Apex DDP + LAMB

# Scaling practices

## 1. Compute:

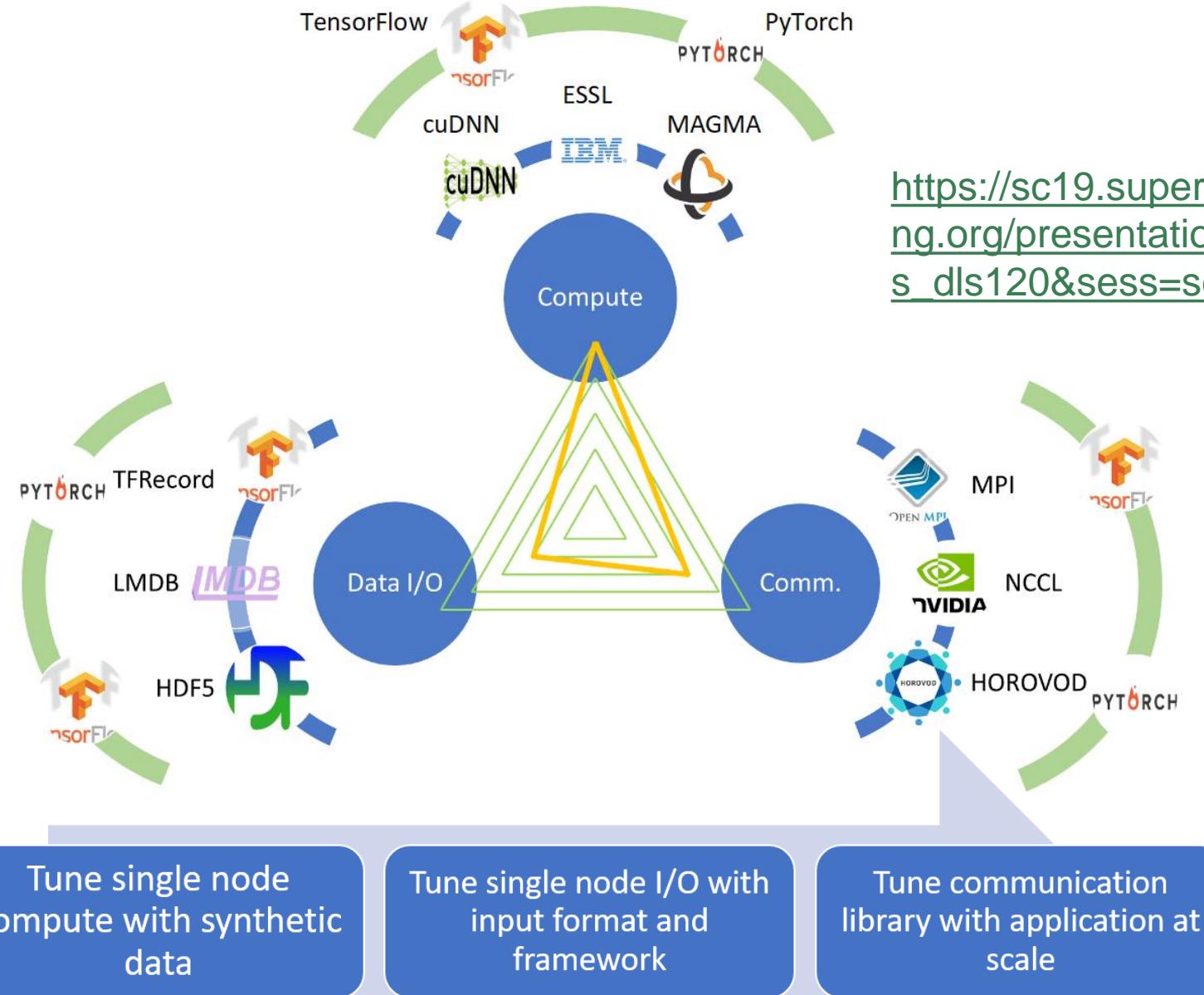
- Tune on single node with synthetic data

## 2. I/O

- Tune on NVMe with input pipelining

## 3. Communication

- Tune at scale with comm. libs



[https://sc19.supercomputing.org/presentation/?id=ws\\_dls120&sess=sess101](https://sc19.supercomputing.org/presentation/?id=ws_dls120&sess=sess101)

# Always checkpointing

- It is relatively straightforward and cheap to checkpoint in DL.
- for data parallel, it is essentially the same as for single GPU.

```
# Save checkpoint
if hvd.rank() == 0:
    state = {
        'model': model.state_dict(),
        'optimizer': optimizer.state_dict(),
    }
    torch.save(state, filepath)
```

```
# Load checkpoint
if hvd.rank() == 0:
    checkpoint = torch.load(filepath)
    model.load_state_dict(checkpoint['model'])
    optimizer.load_state_dict(checkpoint['optimizer'])

# Horovod: broadcast parameters & optimizer state.
hvd.broadcast_parameters(model.state_dict(),
                        root_rank=0)
hvd.broadcast_optimizer_state(optimizer,
                            root_rank=0)
```

# Scaling considerations: Compute

- Use Tensor Cores (2 ~ 4x) 15 TFLOPS(FP32) vs 120 TFLOPS(FP16)

- PyTorch: NVIDIA Apex plugin <https://github.com/NVIDIA/apex>

```
# Added after model and optimizer construction
model, optimizer = amp.initialize(model, optimizer, flags...)
# loss.backward() changed to:
with amp.scale_loss(loss, optimizer) as scaled_loss:
    scaled_loss.backward()
```

- TensorFlow:

```
#Enable TF-AMP graph rewrite:
os.environ["TF_ENABLE_AUTO_MIXED_PRECISION_GRAPH_REWRITE"] = "1"
#Enable Automated Mixed Precision:
os.environ['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1'
```

- Verify on Tensorcore:

```
#Turn off Tensorcore:
os.environ["TF_DISABLE_CUDNN_TENSOR_OP_MATH"] = "0"
#nvprof: tensor_precision_fu_utilization to show TC utilization
```

# Scaling considerations: Compute

- Use XLA (~ 1.5x for ResNet50)

```
#Enable XLA:  
session_config.graph_options.optimizer_options.global_jit_level =  
tf.OptimizerOptions.ON_1
```

- Tune cuDNN algorithms (e.g. 7 implementations for conv)

```
#TensorFlow  
os.environ['TF_CUDNN_USE_AUTOTUNE'] = '1'  
#PyTorch  
torch.backends.cudnn.benchmark = True
```

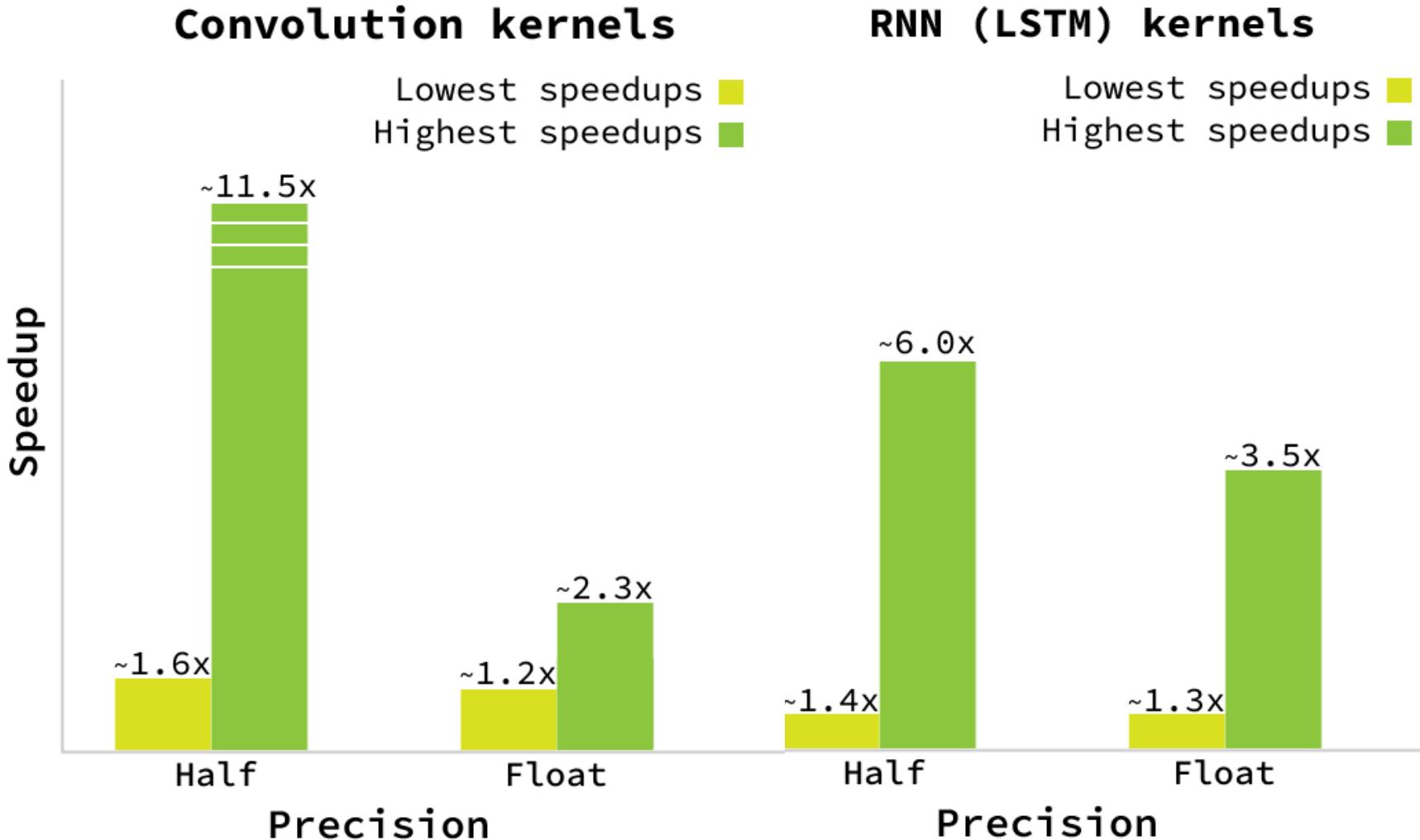
- Know your kernels (optimal scheduling policy)

# Scaling considerations: Compute

- Benchmark kernels

- CNN:  
kernel size, # of kernels,  
etc.

- RNN:  
batch size, timesteps,  
etc.

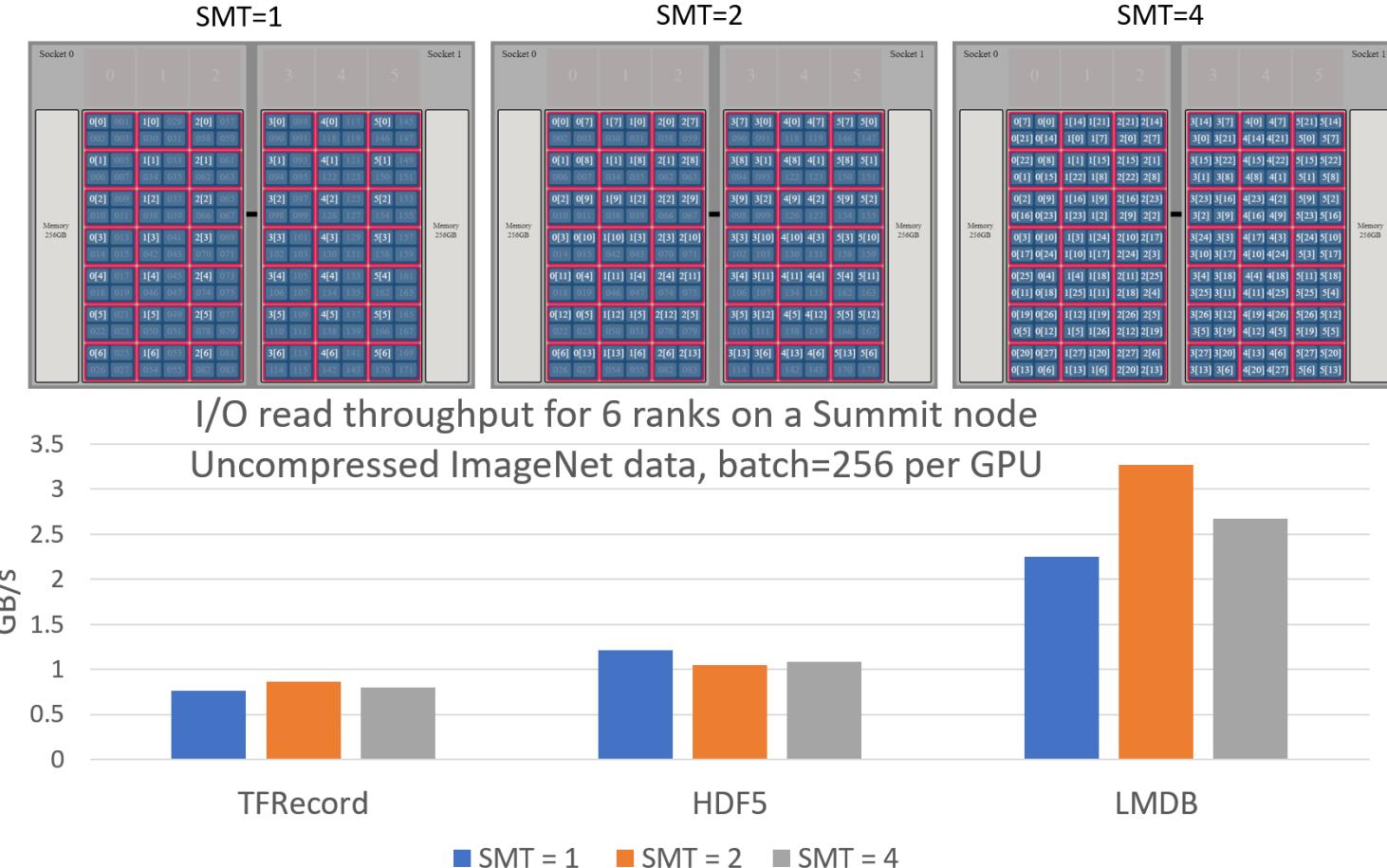


# Scaling considerations: I/O

- Use node local NVMe

Device \ Bandwidth	Full system run
GPUs processing	$3*224*224*4*$ $1200*6*4608/10**12$ $\sim 20 \text{ TB/s}$
GPFS reading	2.5 TB/s
NVMe reading	$6 \text{ GB/s} * 4608 \sim 27 \text{ TB/s}$

- Use LMDB input format



# Scaling considerations: I/O

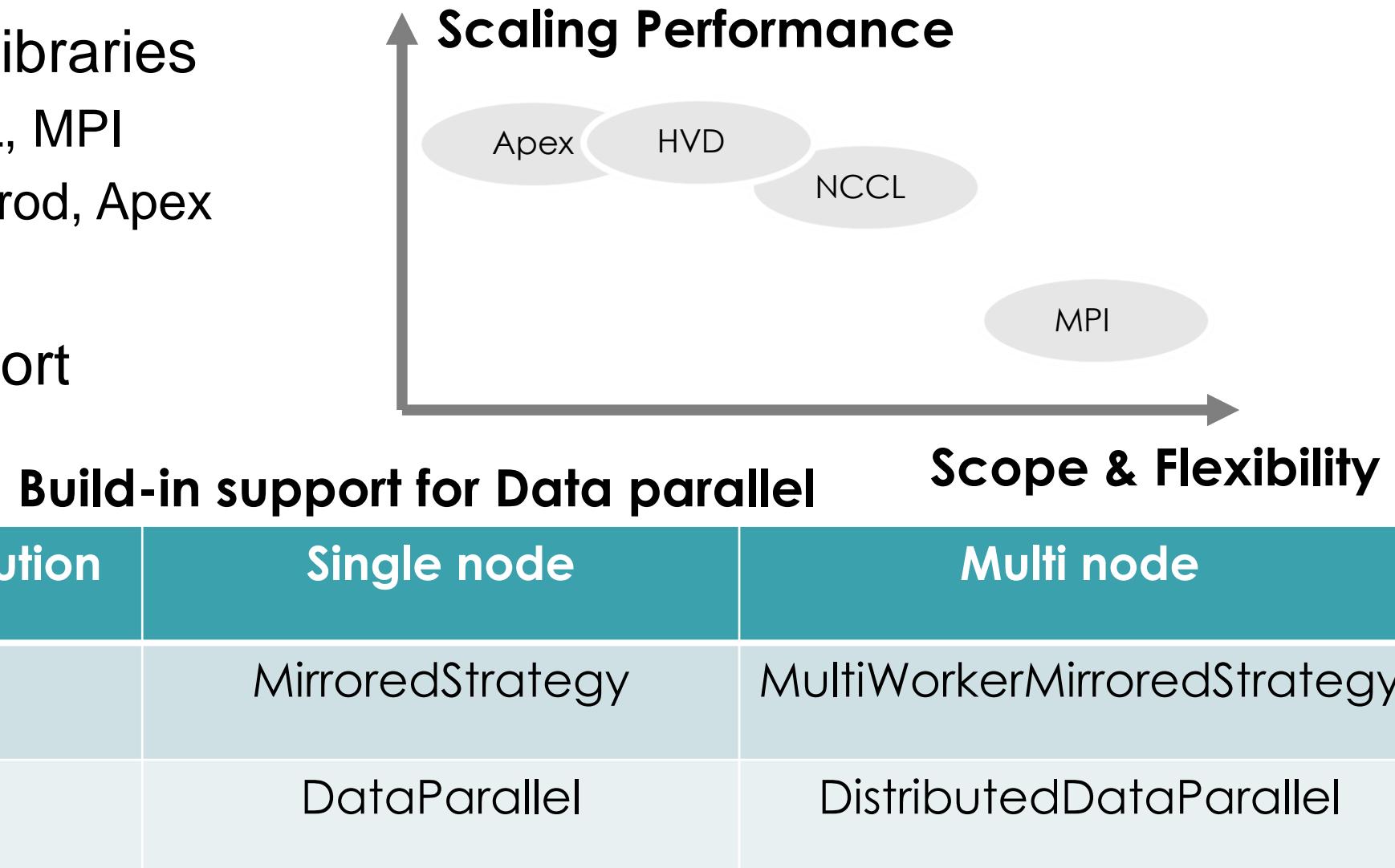
- CPU affinity settings (for pre-processing OPs on CPU)
  - Correct binding (numactl + OMP\_PLACE):

```
# for SMT4
case $((PMIX_RANK%6)) in
[0])
export PAMI_IBV_DEVICE_NAME=mlx5_0:1
export OMP_PLACES={0:28}
numactl --physcpubind=0-27 --membind=0 $APP
;;
[1])
export PAMI_IBV_DEVICE_NAME=mlx5_1:1
export OMP_PLACES={28:28}
numactl --physcpubind=28-55 --membind=0 $APP
....
```

- Use pre-processing pipeline [https://www.tensorflow.org/guide/data\\_performance](https://www.tensorflow.org/guide/data_performance) : staging, prefetch, parallel\_interleave, etc

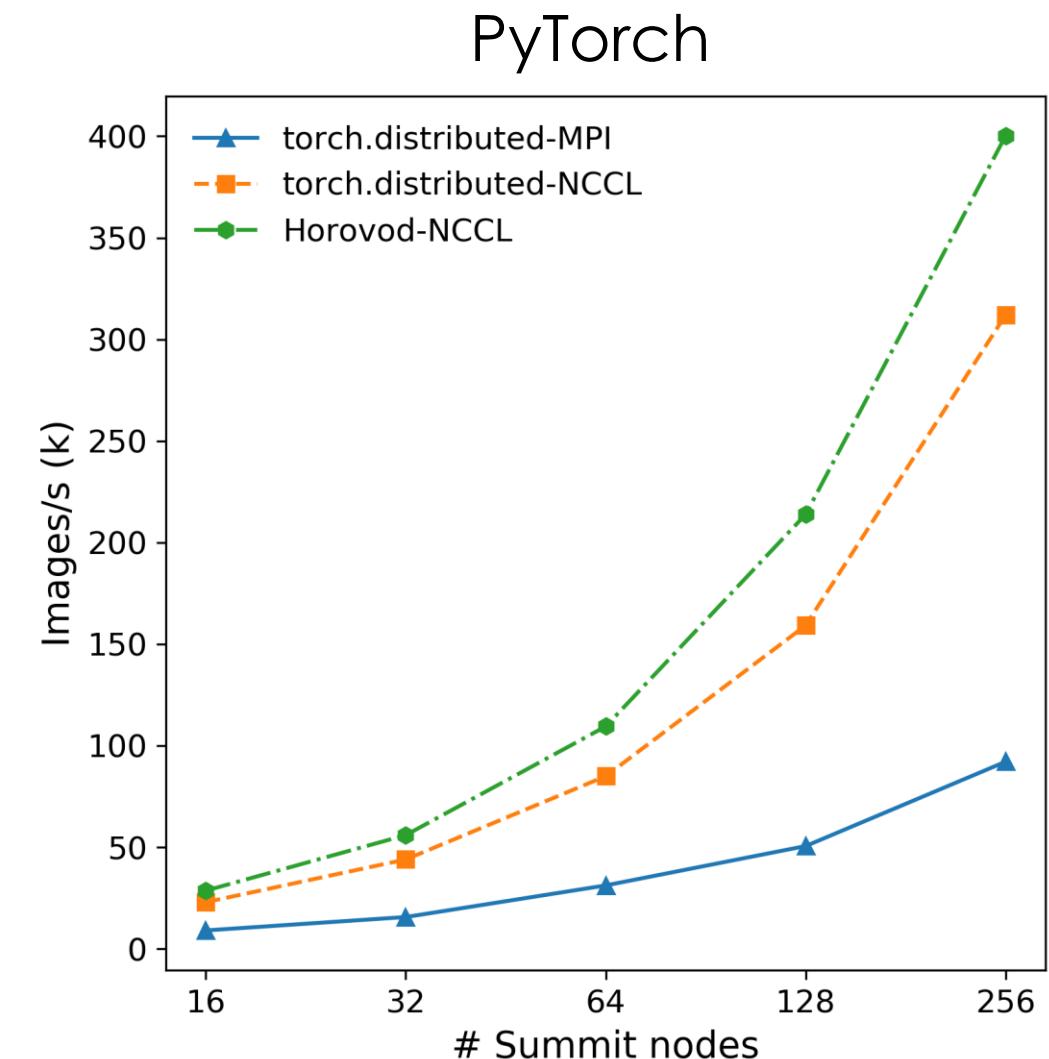
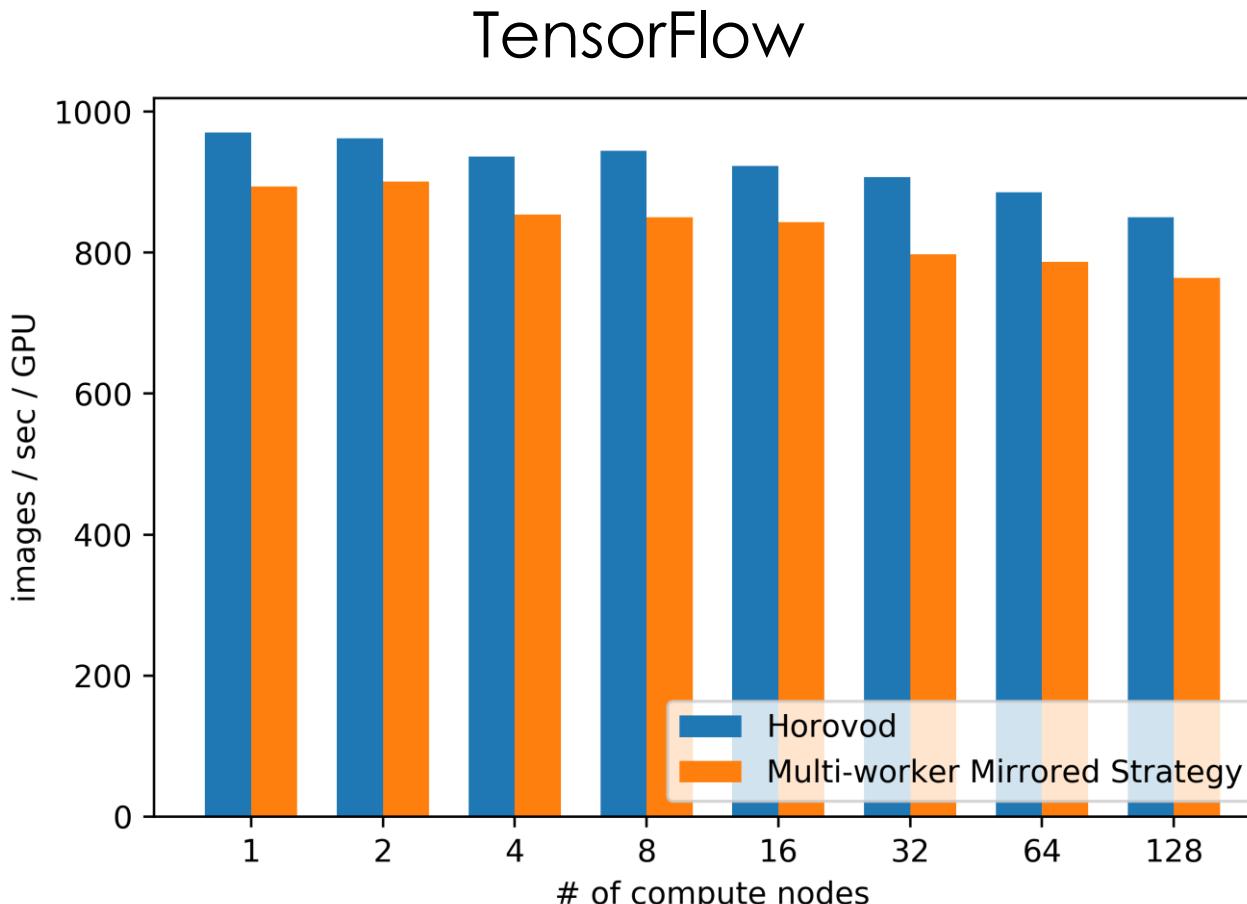
# Scaling considerations: Communication

- Communication libraries
  - Low level: NCCL, MPI
  - High level: Horovod, Apex
- Framework support



# Scaling considerations: Communication

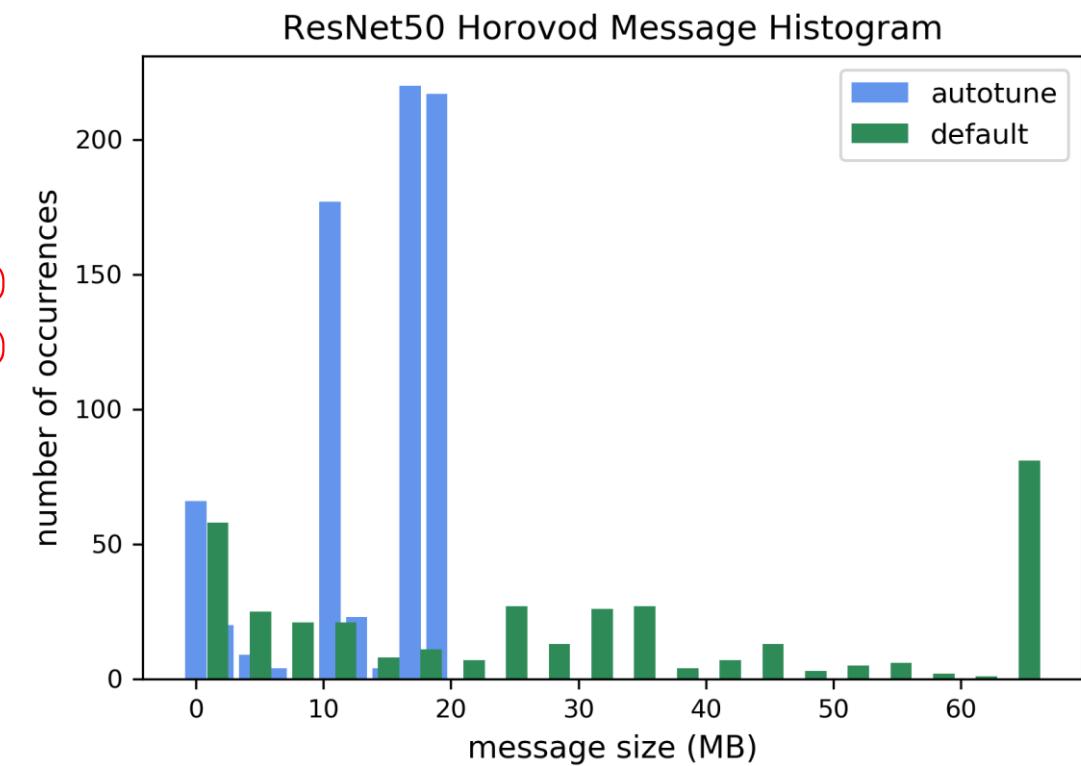
- Use Horovod with NCCL backend



# Scaling considerations: Communication

- Tune Horovod parameters
  - Key knobs: HOROVOD\_CYCLE\_TIME, HOROVOD\_FUSION\_THRESHOLD

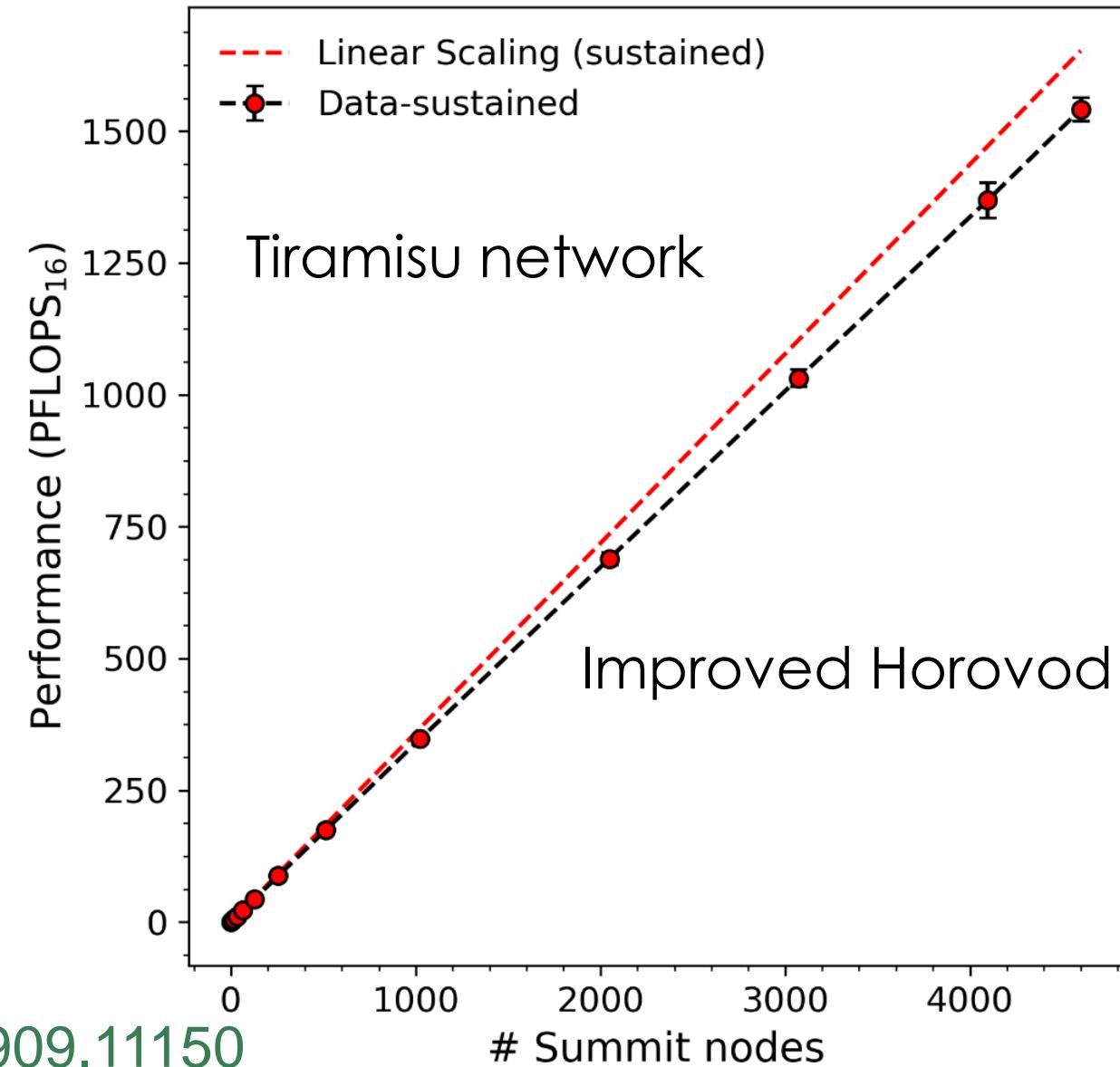
```
#Horovod autotuner
export HOROVOD_AUTOTUNE=1
export HOROVOD_HIERARCHICAL_ALLGATHER=0
export HOROVOD_HIERARCHICAL_ALLREDUCE=0
export NCCL_DEBUG_SUBSYS=COLL
```



# Full Summit Scaling: A Material Science App

- Sustained performance: 1.5 EFLOPS (FP16)
- 93% scaling efficiency at 4600 nodes

Peak Performance: **2.1** EFLOPS (FP16)

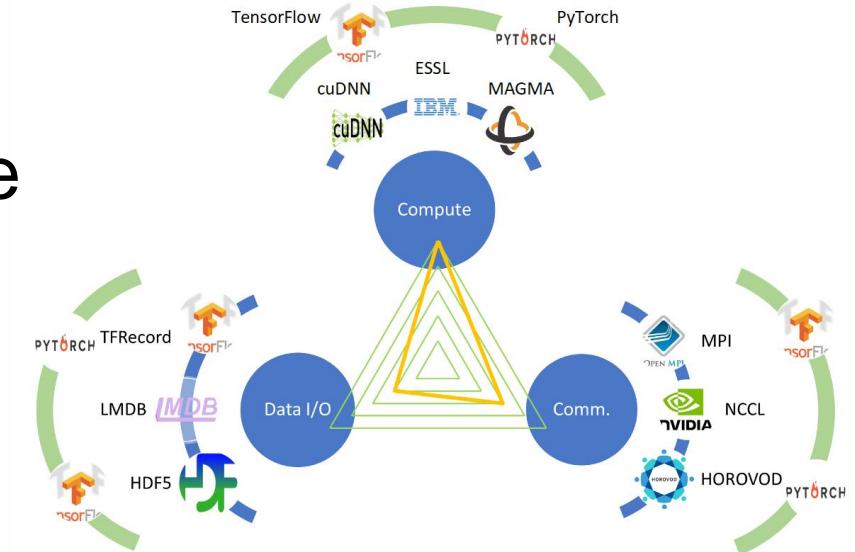


# Conclusion

- Summit is ideal for ML/DL applications
- ImageNet training with ResNet50 can achieve 87% scaling efficiency up to 1024 nodes
- BERT per-training on Wikipedia corpus in about 1 hour on 256 nodes
- Multi-node multi-GPU PCA, Kmeans, SVM, XGBoost etc with Rapids, SnapML, and pbdR

For more information:

- Scaling up DL on Summit webinar: [slides recording](#)
- <https://code.ornl.gov/olcf-analytics/summit/distributed-deep-learning-examples>
- <https://github.com/benjha/nvrapids.olcf>
- <https://code.ornl.gov/olcf-analytics/summit/rapids>



Thank You!

# More resources

- More examples (most run on Summit with minimum changes)
  - Horovod examples:  
<https://github.com/horovod/horovod/tree/master/examples>
  - Nvidia DL examples: <https://github.com/NVIDIA/DeepLearningExamples>
- Official documentations
  - TensorFlow data best practices:  
[https://www.tensorflow.org/guide/data\\_performance](https://www.tensorflow.org/guide/data_performance)
  - TensorFlow distributed training:  
[https://www.tensorflow.org/guide/distributed\\_training](https://www.tensorflow.org/guide/distributed_training)
  - PyTorch data parallel:  
[https://pytorch.org/tutorials/intermediate/ddp\\_tutorial.html](https://pytorch.org/tutorials/intermediate/ddp_tutorial.html)